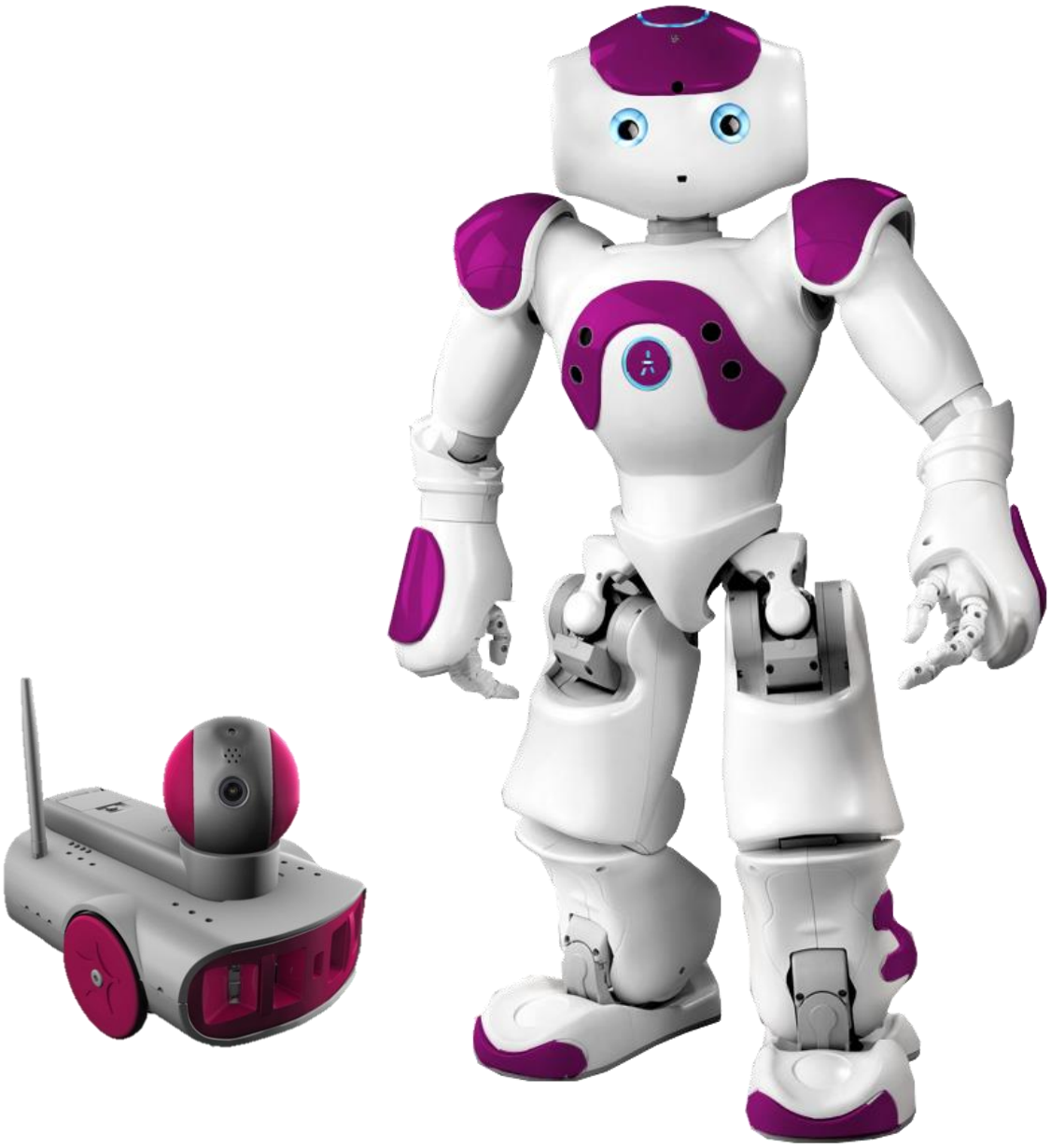
ARTS ET MÉTIERS ParisTech
Paris

**CER PARIS 19/02/2013**



# DESARROLLO DE MÓDULOS DE VISIÓN PARA LOS ROBOTS NAO Y EMOX


Del ARROYO ÁLVAREZ, LUIS
CANO GUEREÑA, DANIEL

Tutor de proyecto : René BOULAIRE

# SUMARIO

# Presentación y objetivos

En el marco del desarrollo de un nuevo laboratorio de robótica y mecatrónica la ENSAM ha comprado varios robots. Gracias a nuestro proyecto, hemos participado en su construcción. Nuestra investigación se ha centrado en los dos robots: en primer lugar el robot humanoide Nao de Aldebaran Robotics, y más tarde la plataforma Emox de Awabot. Ambos robots están concebidos para su interacción directa con humanos: Nao con un aspecto humanoide de adapta de manera más natural, e Emox con una apariencia de animal de compañía. Además, las capacidades particulares de cada robot han determinado nuestra investigación.

Por una parte, Nao, está particularmente adaptado a la ayuda a personas de movilidad reducida como son los proyectos de la empresa en el trato de niños con problemas de autismo. Po resta razón, hemos decidido que la aplicación de Nao sea el apoyo a personas mayores en el momento de la toma de medicamentos gracias a códigos implantados sobre las cajas de tal manera que el robot sea capaz de reconocerlas y facilitársela en el momento oportuno.

Por otra parte, Emox es un robot mucho más ágil que Nao, por lo que hemos decidido orientar en esta capacidad nuestro estudio. Nuestra idea es que el robot sea capaz de manera autónoma de orientarse hacia distintas habitaciones o lugares dentro de una casa inteligente u oficina. Para ello, creamos un camino codificado por colores de tal manera que el robot es capaz de distinguir el camino que recorre y así toma las decisiones adecuadas (acelerar, frenar, tomar un desvío, etc.) para llegar al destino indicado.

En resumen, centraremos nuestro proyecto en la lectura de códigos externos para Nao, y en el desplazamiento y su codificación para Emox.

# 1. Introducción

## 1.1.Estado del arte y trabajos precedentes

### 1.1.1. Nao

Nao es un robot humanoide de unos 57cm de altura con la apariencia de un niño, desarrollado por Aldebaran Robotics, marca de origen francés actualmente absorbida por una firma japonesa llamada Japan's Softbank.

Programable en varios lenguajes como C++, Python, Urbi, .NET, Java y Matlab. Equipado con un procesador Intel ATOM 1,6 Ghz que funciona con un núcleo Linux y el software dedicado propio Aldebaran Naoqi.

Nuestro proyecto, centrado en la visión, se lleva a cabo con dos cámaras posicionadas en la cabeza (en la frente y la boca) capaces de tomar 30 imágenes por segundo a una calidad de 920p.

Encontramos proyectos anteriores que nos ayudan en nuestra investigación en la propia biblioteca de trabajos de la empresa en developer.aldebaran-robotics.com.

Aquí encontramos un proyecto que guarda mucha relación con el nuestro que se encarga de la descodificación de códigos DataMatrix. La última versión de este proyecto se descarga en un fichero proprio de Choregraphe, el software propio del robot con el que diseñamos el árbol de decisiones (DataMatrix-1.0.crg). Para la lectura usa una librería gratuita llamada libdtmx. Este proyecto ideado para la versión 1.12 de Nao, desgraciadamente no funciona para la nueva actualización, la 1.14.

### 1.1.2. Emox

Emox es un pequeño robot con ruedas de unos 23m de largo, 15 cm de ancho y 16 de alto. En el momento de la investigación el robot se encuentra en desarrollo, por lo que trabajamos con un prototipo de lo que será comercializado en principio, en marzo de 2013 en el salón Innorobo que se llevará a cabo en la ciudad de Lyon, Francia.

En la imagen podemos ver la versión con la que trabajamos a la izquierda, y a la derecha la versión que saldrá a la venta. El único cambio remarcable es la estructura exterior ya que el prototipo está en estado avanzado y viene equipado con todos los componentes necesarios.

Adjuntamos también un video promocional del mismo para entender de manera directa el proyecto Emox:

http://www.youtube.com/watch?v=4eBqVn3uS5w

De la misma forma que encontramos en Nao, para Emox también disponemos de un espacio web donde encontramos proyectos creados por otros usuarios, un foro de dudas e información técnica: http://awabot-redmine.herokuapp.com/.

### 1.1.3. Códigos en 1D et 2D

Hoy en día la lectura de códigos está muy desarrollada en los smartphones, es por esto por lo que encontramos una larga lista de librerías y aplicaciones destinadas a ello que nos pueden servir en el proyecto. Por ello es necesario hacer unan intensa búsqueda analizando los pros y contras de cada una para decidirnos por una o dos. Nos basamos en el lenguaje de programación, la facilidad de modificación e de inserción en nuestros módulos, la facilidad de descarga y las características multiplataforma.

Además, debemos elegir qué códigos usar ya que tenemos varias opciones: Códigos de barras, Datamatrix, QR o NaoMark.

Descartamos los de barras por su escaso tamaño de información, y los Datamatrix por su necesidad de una cámara potente para su lectura, y por su escaso uso. Los códigos QR los encontramos por todas partes, aún en constante desarrollo por lo que creemos que es la mejor opción, la más internacional y con la que encontraremos más apoyo informático. Hablaremos de los NaoMark en su epígrafe correspondiente.

Además, ofrecen la particularidad de superponer logotipos ya que cuentan con un algoritmo que permite compensar errores de imagen. Sirva de ejemplo la imagen siguiente que nos enlaza con http://www.ensam.fr/ :


Finalmente escogeremos los códigos QR y la modificación de la aplicación qrdecoder.

# 2. NAO

## 2.1. Instalación de las herramientas y librerías necesarias en para trabajar con Nao.

Es necesario instalar las librerías que usaremos y los programas. Para ello, nos dirigimos a la web de Aldebaran donde explican, para cada versión del robot, los pasos a seguir:

http://www.aldebaran-robotics.com/documentation/dev/cpp/install_guide.html

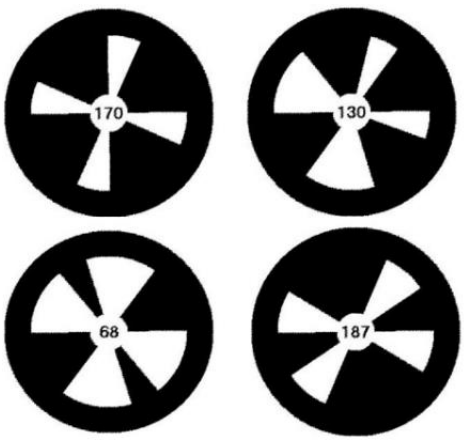
Para usar la aplicación qrdecoder, necesitaremos instalar lo estipulado en http://code.google.com/p/qrdecoder/ , es decir:

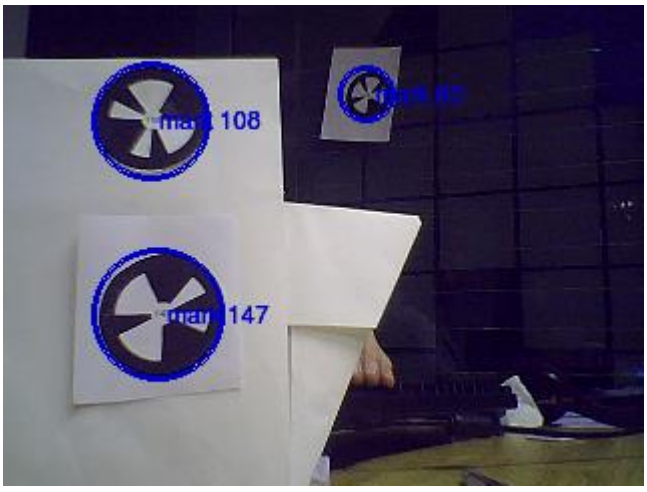
- Python - by the Python Software Foundation
- PyQt4 GUI toolkit by Phil Thompson - GNU GPLv3
- SIP by Phil Thompson - GNU GPLv3
- ZBar by Jeff Brown - GNU LGPLv2.1
- Python Imaging Library (PIL) by Secret Labs AB & Fredrik Lundh

## 2.2. Los códigos estudiados

### 2.2.1. NaoMarks

Existe una codificación propia de Aldebaran. Son códigos redondos capaces de almacenar nada más que números. A pesar de ser propios para el robot, encontramos que la aplicación de lectura aún no está muy bien implementada y hay muchos problemas a la hora de leer el código desde distintos ángulos. Así y dada su bajo uso y por ello poco soporte técnico, decidimos descartarlos ya que la mejora de su lectura no forma parte de nuestro proyecto.



### 2.2.2. Códigos de barras

A pesar de su gran uso en supermercados y grandes superficies, estos códigos apenas pueden guardar unos pocos bytes de información, ocupando gran cantidad de espacio. Descartamos entonces también esta codificación.



### 2.2.3. Códigos QR

Creados en 1994 por Denso-wave, ya existiendo en aquel momento los códigos Datamatrix, la novedad de estos códigos es el almacenamiento de los caracteres japoneses kanji abriendo un mundo enorme de explotación. Así, las ventajas de los QR frente a los Datamatrix son tres:

- **Uso del espacio más eficaz**

La información guardad en una misma área que los otros códigos es mayor, pero con una densidad de puntos suficiente para ser leída por las cámaras que usamos habitualmente en nuestros aparatos.

- **Almacenamiento de caracteres japoneses (kanji)**

Es necesario hoy en día que los caracteres japoneses se tengan en cuenta dada la expansión y el poder tecnológico del país.

- **Sistema de control de pérdida más eficaz:**

Así es, el algoritmo de corrección Reed-Solomon permite tener varios niveles de error (7, 15, 25, 30 %). Como ejemplo el código ya mencionado en el que hemos superpuesto un logotipo en el medio de la imagen, y aun así los decodificadores con capaces de leer el código.

### 2.2.4. Datamatrix

Estos códigos son la competencia directa de los QR. Ideados en 1989 por la empresa estadounidense International Data Matrix, tienen como última versión la lanzada en 2005.

A pesar de tener también la opción de superponer logotipos, es necesario aumentar el espacio del código obteniendo así una mayor densidad de puntos y entonces encontramos el problema de la dificultad de lectura por Nao:

## 2.3.Las librerías y aplicaciones estudiadas

Tras llevar a cabo una ardua tarea de búsqueda e investigación, encontramos posibles librerías y/o aplicaciones de descodificación, a saber:

- Qrdecoder
- Zbar
- OpenCV
- ZXing

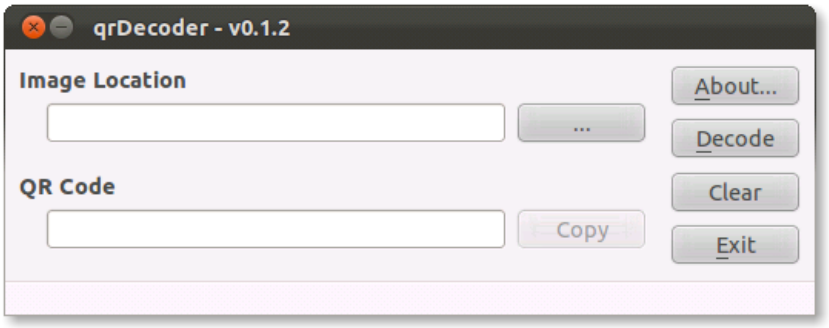
Finalmente nos decantamos por su facilidad de modificación por la aplicación qrdecoder basada en el decodificador Zbar.

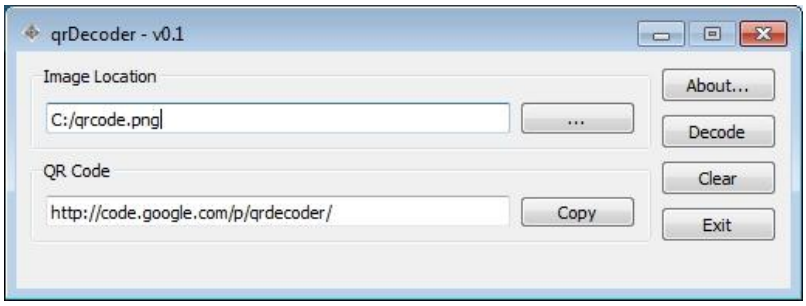
### 2.3.1. qrdecoder

***qrdecoder*** es una aplicación creada por un particular llamada Nicholas Wilde. Es open source y descargable en: http://code.google.com/p/qrdecoder/

La aplicación está ideada especialmente para leer códigos Qr de imágenes en los formatos:

- PNG
- GIF
- TGA
- JPG
- PCX
- BMP
- TIF




Está escrita en python, el cual es un lenguaje muy usado en el mundo de la robótica por su sencillez. Es una de las características que han hecho que la elijamos frente a otras.

#### *2.3.1.1. PyQT4, PIL et SIP*

De estas tres librerías la que nos interesa es esencialmente es PIL. PyQT4 se encarga de la ventana y SIP gestiona los elementos multimedia.

PIL fue creada para tratar imágenes en entornos Python. Con ella obtenemos la información de la imagen (no del código) y la convertimos a blanco y negro (de ahí que podamos colorear como queramos los códigos que no afecta al mensaje guardado)

```
# obtain image data
pil = Image.open(unicode(self.lineEdit_location.text())).convert('L')
width, height = pil.size
raw = pil.tostring()
```

La información obtenida es del estilo:

```
<Image.Image image mode=L size=250x300 at 0x93F18EC>
```

### 2.3.2. ZBar

ZBar es una librería escrita en C, open source, compatible con C++, Python, etc. Permite obtener los mensajes codificados en códigos QR y de barras:

- QR codes
- UPC-E
- Code 39
- EAN-13
- EAN-8
- Interleaved 2 of 5
- UPC-A
- Code 128

Con ella podemos escanear o bien de una imagen fija o de un vídeo. Encontramos distintos tutoriales en :

http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_images_using_the_API

http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_video_using_the_Processor

Nos interesa mucho el escaneo de videos en tiempo real para el proyecto que queremos ejecutar en Nao.

Finalmente escribimos un código en Python que usa esta librería para leer los QR en imágenes fijas. Código testado con éxito en Linux Ubuntu.

#### *2.3.2.1. Usando OpenCV en vez de ImageMagick*

ZBar usa ImageMagick para el tratamiento de imágenes. Dada su poca expansión y la función natural de Nao de manejar OpenCV, nos interesa hacer un cambio en el código actual. Así, los cambios necesarios se explican a continuación en C++:

**Con ImageMagick**

```
// read an image file

Magick::Image magick(argv[1]);

// extract dimensions

int width = magick.columns();

int height = magick.rows();

// extract the raw data

Magick::Blob blob;

magick.modifyImage();

magick.write(&blob, "GRAY", 8);

const void *raw = blob.data();

// wrap image data

Image image(width, height, "Y800", raw, width * height);

// scan the image for barcodes

int n = scanner.scan(image);
```

**Con OpenCV**

```
CvMat*cv_matrix=cvLoadImageM(argv,CV_LOAD_IMAGE_GRAYSCALE);

ImageScanner scanner;

scanner.set_config(ZBAR_NONE,ZBAR_CFG_ENABLE, 1);

int width = cv_matrix->width;

int height= cv_matrix->height;

char*raw = (char*)cv_matrix->data.ptr

Image image(width, height, "Y800", raw, width * height);

int n = scanner.scan(image);
```

Por causas ajenas, debimos frenar el proyecto de Nao para centrarnos en Emox. Así, la parte que haría falta cambiar en Python es la siguiente:

```
# obtain image data
pil = Image.open(img).convert('L')
width, height = pil.size
raw = pil.tostring()

# wrap image data
image = ZBar.Image(width, height, 'Y800', raw)

# scan the image for barcodes
scanner.scan(image)
```

### 2.3.3. OpenCV


OpenCV es la aplicación open source la más usada actualmente. Creada por Intel en 1999 dispone hoy en día de más de 500 funciones diferentes compatibles en GNU/Linux, Windows o Mac OS X. Nao viene con esta librería implantada para, por ejemplo, la detección de rostros.

El problema que hemos encontrado es que tras hacer la actualización de versión de sistema interno de Nao, encontramos un cambio de OpenCV 2.1 a 2.2, es decir, del OpenCV clásico a una nueva versión OpenCV2. Así, todos los códigos escritos anteriormente dejaron de funcionar ya que el cambio es importante y necesita de muchos cambios en las llamadas a las funciones externas para ejecutarlos. Como decimos anteriormente, debimos dejar de lado el proyecto Nao así que estos son los cambios a realizar por los alumnos que retomarán nuestro trabajo a los cuales les dejamos todo el proceso lo más detallado posible.

## 2.4.Integración de las aplicaciones en Nao

Encontramos dos métodos para implantar nuestros propios módulos en el robot:

- Creando módulos C++.
- Con un proyecto de Choregraphe.

### 2.4.1. Por consola

Tomando como base el código *qrdecoder.pyw*, lo reprogramamos con el fin de poder utilizarlo para el reconocimiento de códigos QR en Nao. Recordamos que el original, sirve para leer una imagen almacenada en nuestro sistema a través de la interfaz. Lo que queremos es que Nao haga una foto y sea ésta imagen la que descodifique son pasar por la interfaz. En esta línea:

```
pil = Image.open(unicode(self.lineEdit_location.text())).convert('L')
```

es `self.lineEdit_location.text()` quien devuelve la localización de la imagen. Lo que haremos será que esa dirección de memoria sea la que envíe Nao cuando tome la foto. Simulamos a continuación el funcionamiento con una dirección de fichero pero que será igualmente válida con una dirección de memoria:

```
#=-=-=-=-=-=-=-=-=-=-=-=-=-=- MAIN  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

print "\n Quelle image voulez-vous décoder? \n"
print "\n"

imagen = raw_input (">")
#imagen = '/home/luis/PJE_linux/qrdecoder/qrdecoder-0.1.2/test/baboon.jpg'

decodeImage(imagen)
```

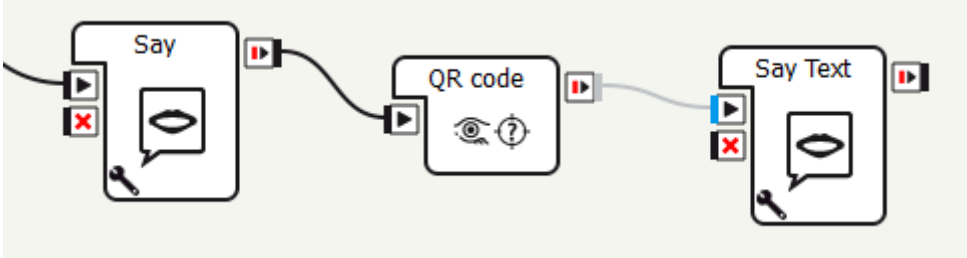
```
#=-=-=-=-=-=-=-=-=-=-=-=-=-=- DEF  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

def decodeImage(img):

    scanner = ZBar.ImageScanner()
    scanner.parse_config('enable')

    pil = Image.open(img).convert('L')
    width, height = pil.size
    raw = pil.tostring()

    pil.show()
```

Tras los cambios necesarios el código está preparado para enviar el mensaje contenido. Quien se encarga de la descodificación es:

```
# scan the image for barcodes
    scanner.scan(image)
    # extract results

    for symbol in image:
      if symbol:
          print "Image contain a code."
      else:
          print "Image does not contain a QR code."

    for symbol in image:

      print symbol.type

      # do something useful with results
      if symbol.type is not symbol.QRCODE:
          print "Image does not contain a QR code."
          return

      print "L'information du code est: \n ", symbol.data, "\n"

    # clean up
    del(image)
```

### 2.4.2. Choregraphe

Después de estudiar el proyecto llamado DataMatrix (encontrado en developer.aldebaran-robotics.com. ) que lee ese tipo de códigos, decidimos seguir sus pasos. Quedaría:




Modificamos el script que recupera la imagen del robot y la analiza con libdtmx. Para ello haría falta cambiar las líneas siguientes:

```
   # Get image from Nao's camera

self.cam.subscribe("dmtx", kQQVGA, kRGBColorSpace, 5); # Warning, use
QQVGA (160x120px) otherwise it's horribly long

 img = self.cam.getImageRemote("dmtx");

 self.cam.unsubscribe("dmtx");
```

```
# Decode the image

decodeImage(img) ;  ## au lieu de self.dm.decode(img[0], img[1],
                        buffer(img[6]));

## On rappelle que  img = self.cam.getImageRemote("nom_image");

    # Parse results

    result = list(symbol.data);

    ##pour remplacer result = list(self.dm.message(i) for i in range(1,
    self.dm.count()+1));
```

y añadir la librería destinada a leer códigos QR linkando las librerías de manera estática. Mantuvimos una conversación con el creador de ese proyecto que fue quien nos dijo que esos son los pasos a seguir:

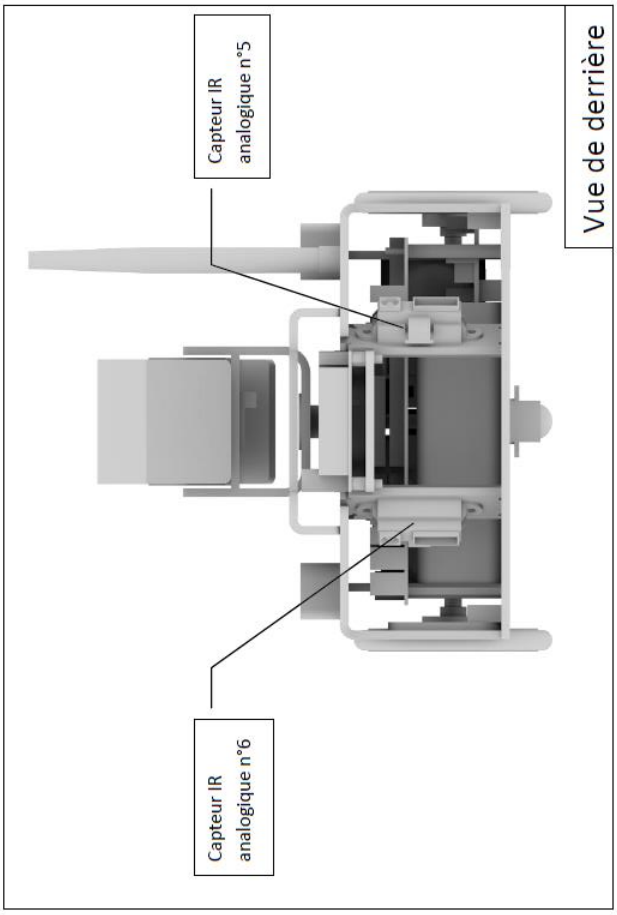
```
def onLoad(self):
    from ctypes import *;
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libdmtx.so');
```

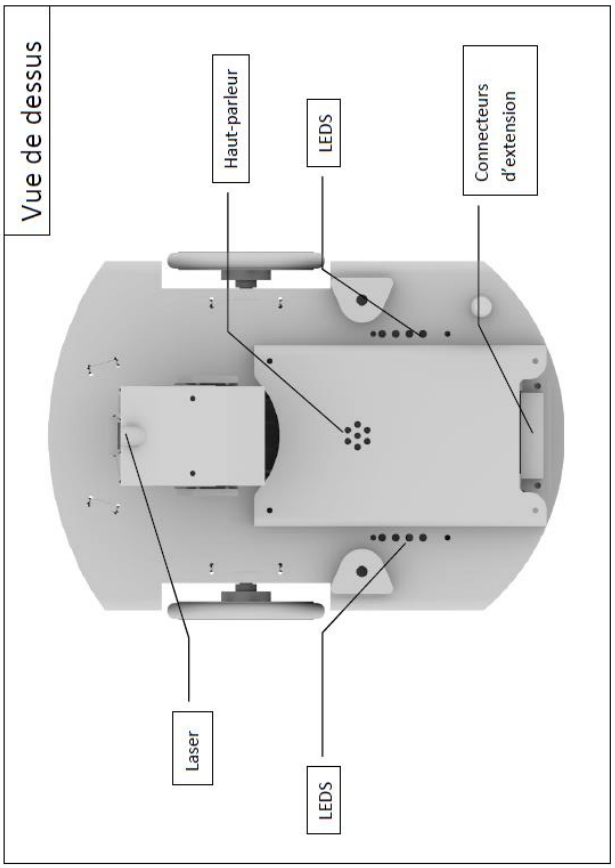
```
def onLoad(self):
        import sys, os, ZBar, Image, sip, string;
    from ctypes import *;
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libdmtx.so');
    cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/CORE_RL_magick_.lib');
        cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) +
'/lib/CORE_RL_Magick++_.lib');
        cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) +
'/lib/CORE_RL_wand_.lib');
        cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libZBar-
0.def');
        cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/libZBar-
0.lib');
        cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId) + '/lib/X11.lib');
```

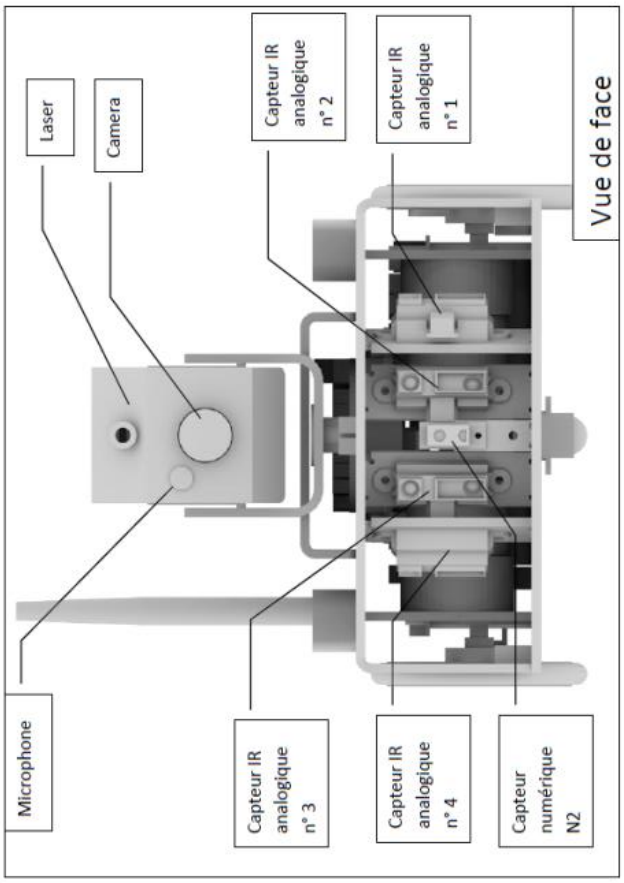
# 3. EMOX

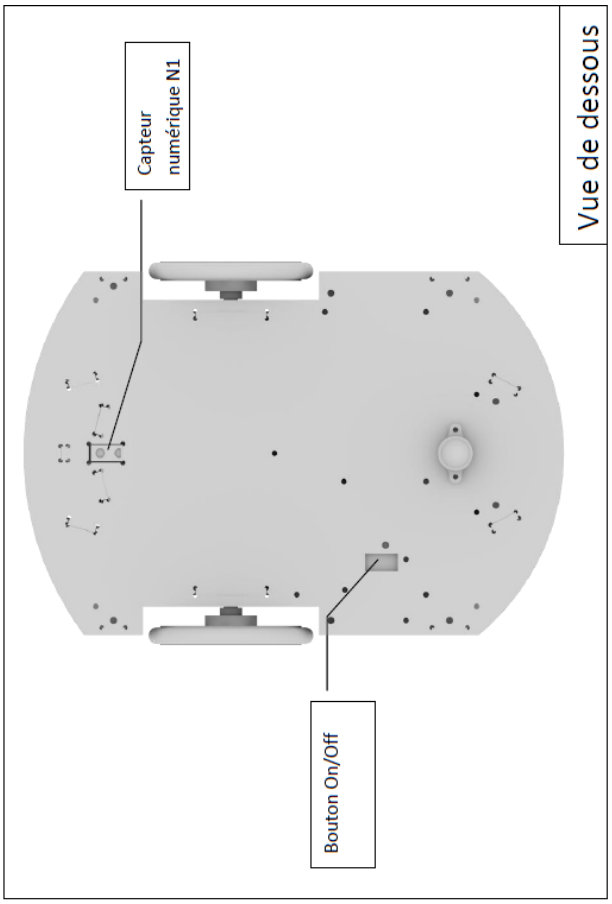
## 3.1. Descripción técnica

Durante el proyecto, trabajamos con un prototipo del robot Emox. El robot se encuentra en desarrollo y se espera una nueva versión para Marzo 2013. Sus características serán muy parecidas a las actuales



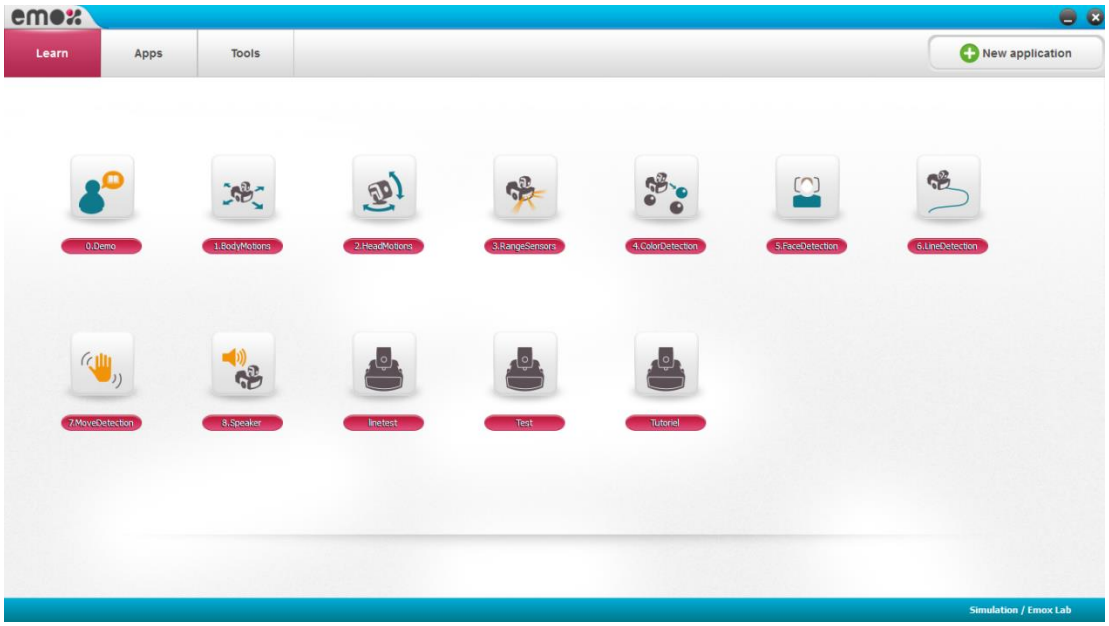






## 3.2. Conexión y programación

Para acceder al robot, utilizamos el software proporcionado por Awabot: Emox Soft. Debimos esperar a la versión 1.0.1 puesto que con la 1.0.0 no nos era posible establecer conexión alguna entre el pc y el robot.



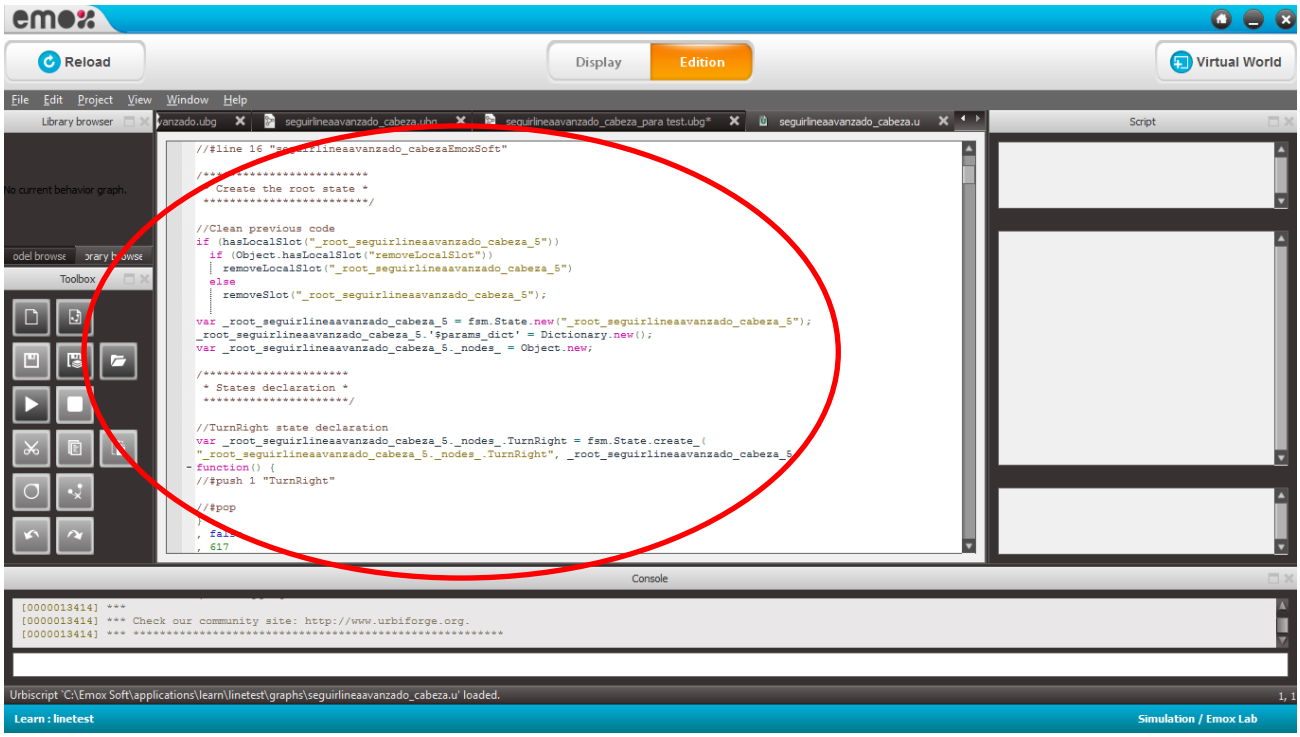
Gracias al Emox Soft, es posible establecer la conexión entre el equipo y el robot vía wi-fi. Además permite crear, editar y lanzar aplicaciones. Dichos programas, pueden ejecutarse en el robot real, o bien, mediante un entorno virtual. Este entorno, incorpora elementos con los que probar el comportamiento del robot, tales como pelotas de colores, un laberinto, una cara móvil, líneas en el suelo, etc.

Emox funciona con Linux y se programa con el lenguaje Urbi. Este lenguaje de programación se compone de UObject del C++ y del lenguaje dirigido por eventos urbiscript. Urbi fue creado en 1999 en París y pertenece a Gostai desde 2006.

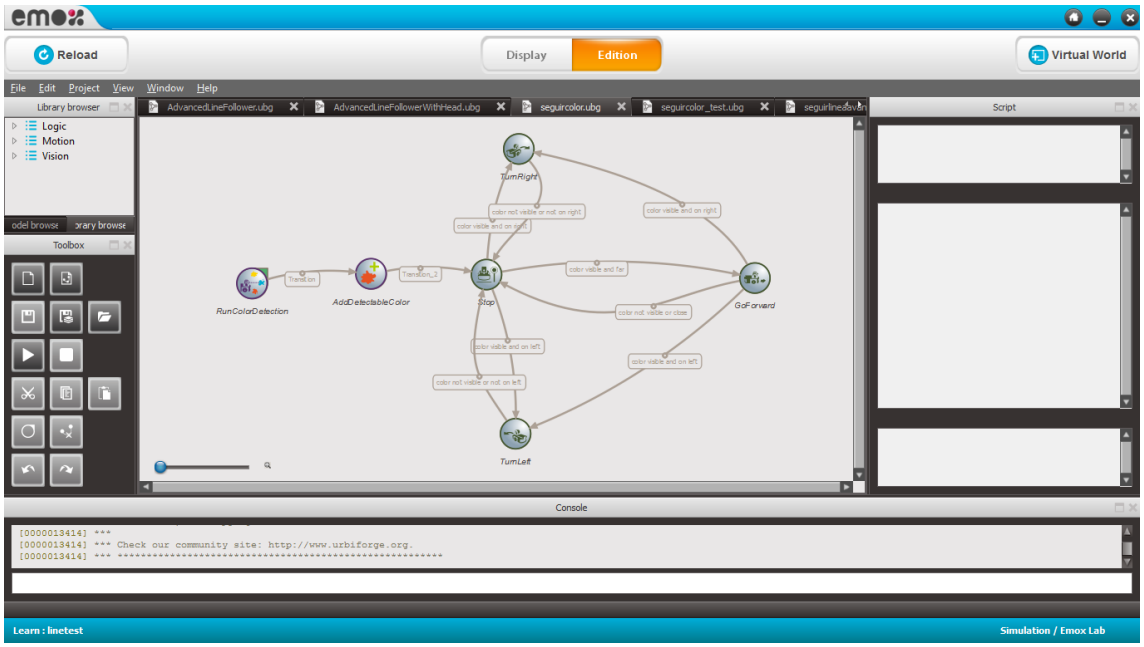
Para programar a Emox, utilizamos pues Emox Soft por lo que trabajamos con nuestras aplicaciones directamente en el entorno de Awabot con el que luego las ejecutamos. Los módulos creados se ejecutan en el ordenador por lo que el robot, envía los datos de los sensores al ordenador que decide y envía las instrucciones a Emox.

Existen 3 formas de programación:

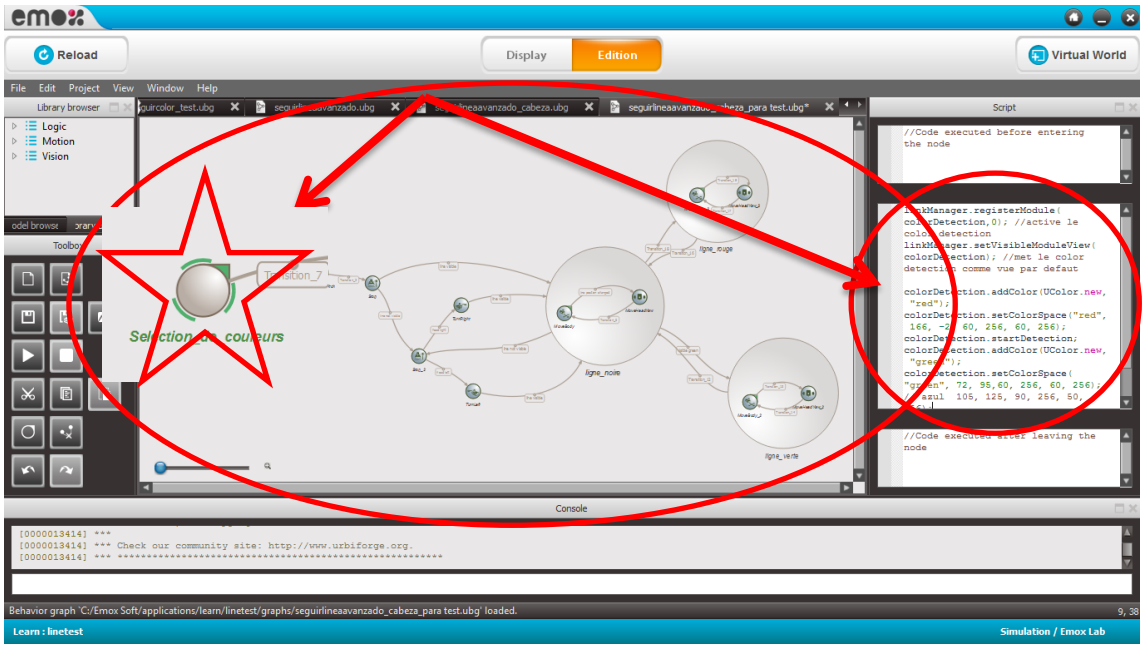
- Programación en script (SDK)




- Programación gráfica:

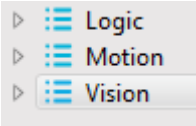



- Programación mixta:



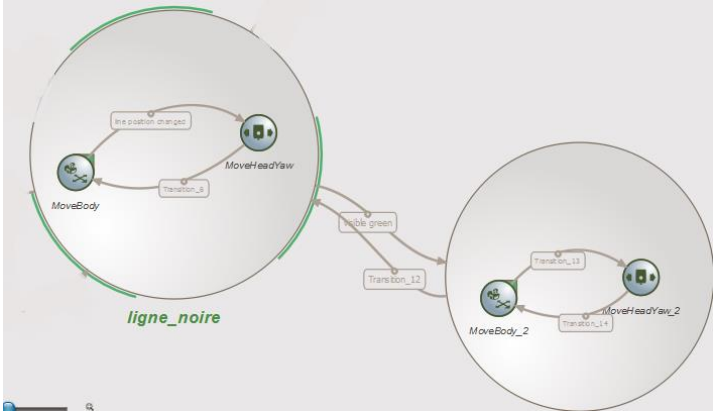

El primer método es complejo por la dificultad del código Urbi y porque la documentación del SDK y de la API aún no están completas

La programación gráfica es intuitiva y fácil de utilizar pero puede ser insuficiente para desarrollar aplicaciones más complejas. Disponemos de 3 bibliotecas principales:




Estas contienen los nudos de programación:


Que a su vez, serán unidos por transiciones que podremos programar:

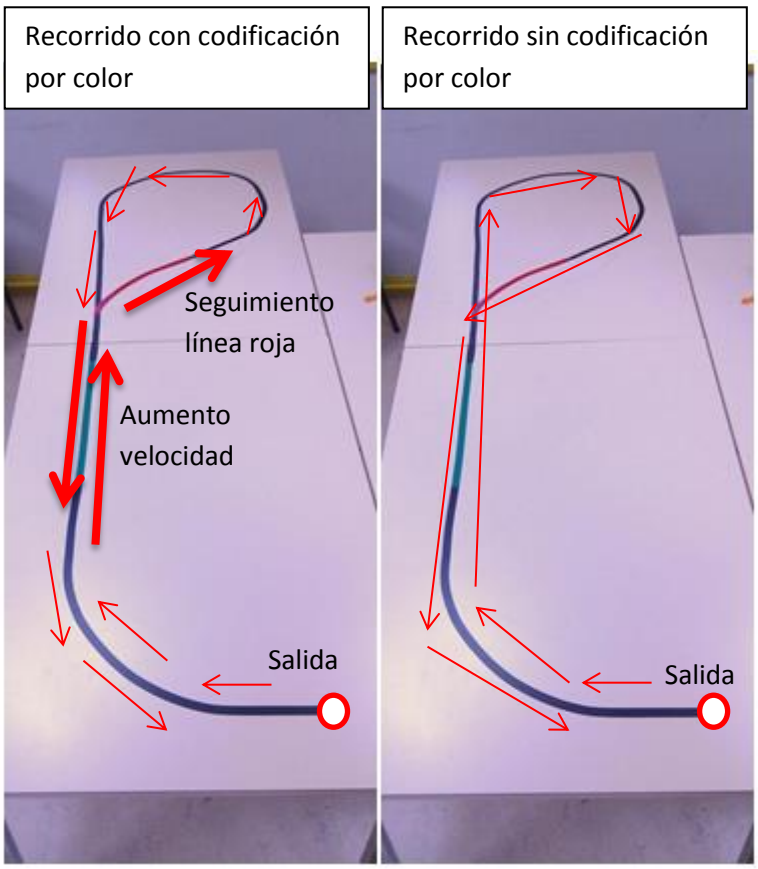



Finalmente, podemos concluir que el mejor método de programación es una combinación entre la interfaz gráfica y la API. De esta forma, podemos aprovechar las ventajas de la programación por eventos y crear nuestros propios nudos, más complejos, mediante el srcipt.

## 3.3. Seguimiento de una línea codificada con colores

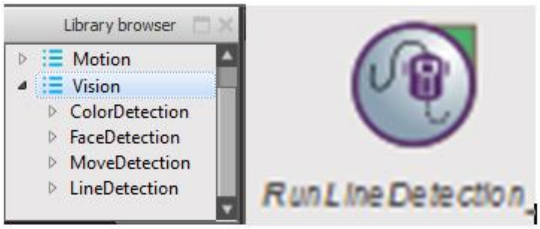
Nuestro objetivo es que Emox sea capaz de seguir una línea situada en el suelo, modificando su comportamiento en función del color de dicha línea.

En este caso, vamos a simular un recorrido con una línea verde y otra roja. En la línea verde, el robot deberá acelerar, mientras que una línea roja en una bifurcación indica al robot que ese es el camino a seguir.



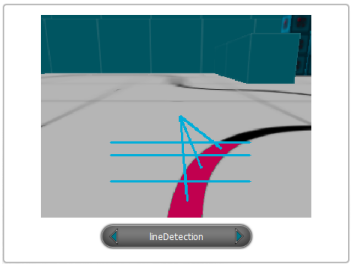

### 3.3.1. Seguimiento de una línea

Para seguir la línea empleamos un módulo existente creado por Awabot.

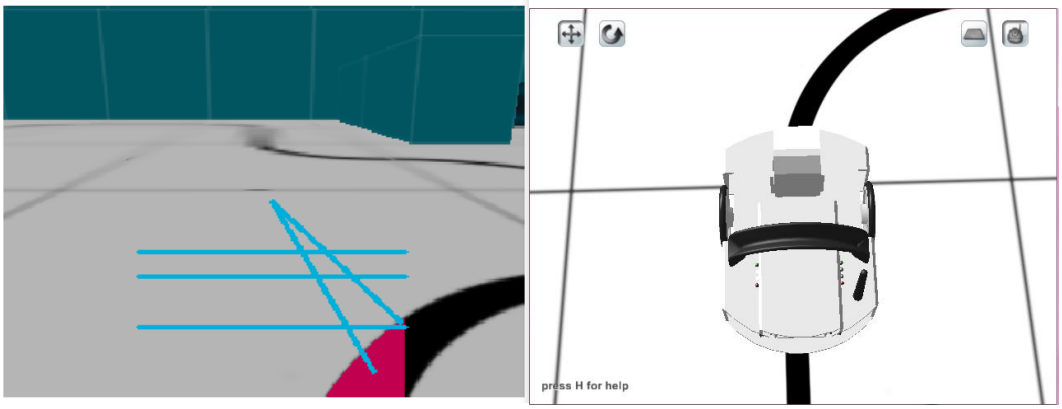


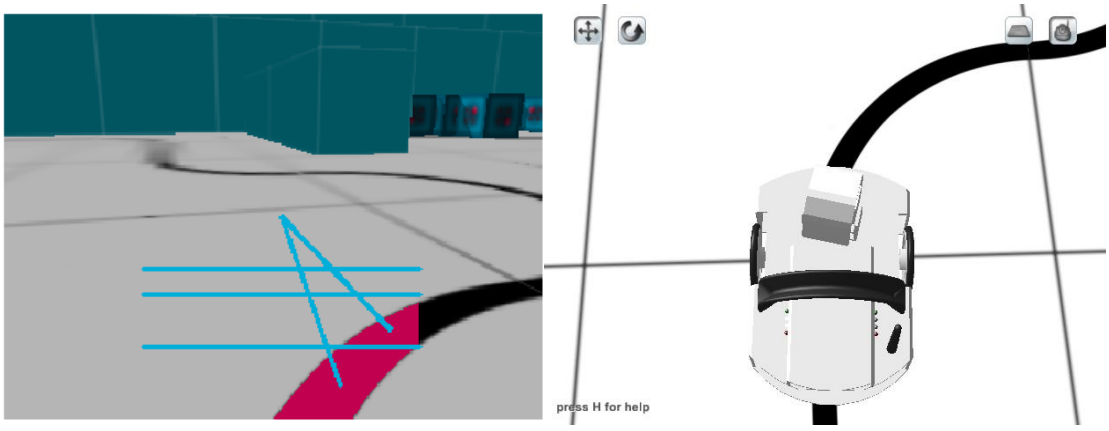

Este módulo puede perder la línea en algunos casos porque lo que lo modificamos para mejorar su comportamiento. Para ello modificamos los parámetros de forma que las correcciones de posición no sean tan bruscas como por defecto, mejorando el seguimiento de la línea sin perderla.

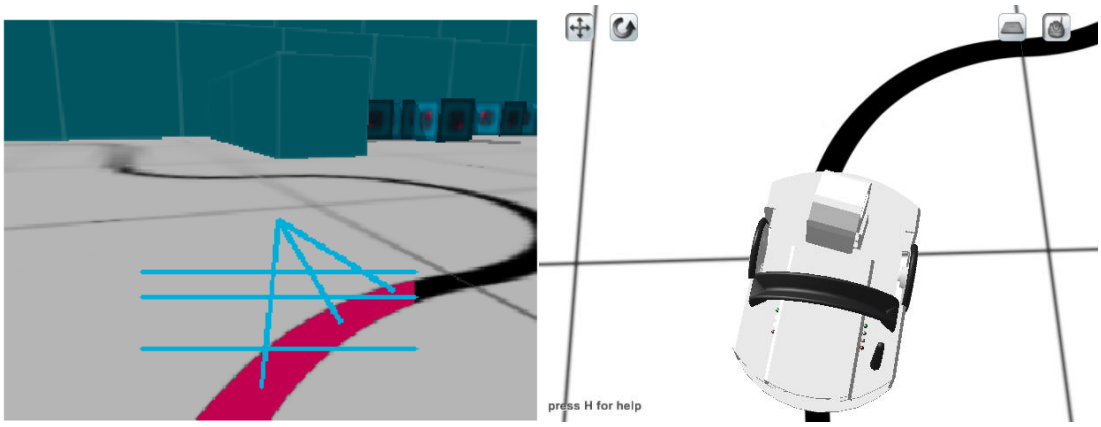
Con Line Detection, la imagen se divide en 3 partes. Así, podemos evaluar la posición de la línea a 3 distancias diferentes, de forma que podemos aplicar un coeficiente diferente a cada zona para seguir mejor el camino.




Para hacer el seguimiento, la cabeza se orienta con la línea y posteriormente, el chasis del robot, se ajusta a la posición de la cabeza. Estos ajustes se hacen de forma alternativa y manteniendo una velocidad constante de 20 cm/s. Los ciclos son tan cortos que lo percibimos como un movimiento continuo.



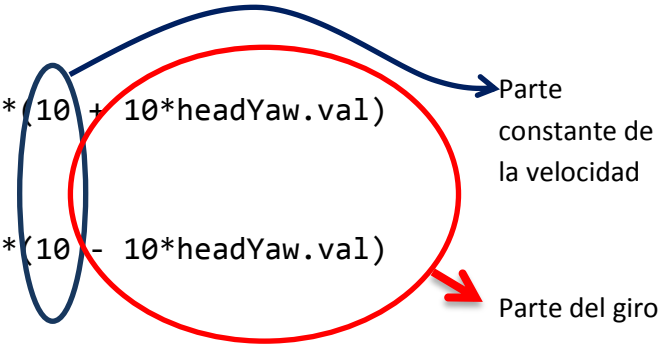



Cabeza sigue la línea:

```
headYaw.val = headYaw.val - (0.1 *
                lineDetection.getZone(0).x + 0.15 *
                lineDetection.getZone(1).x + 0.2 *
                lineDetection.getZone(2).x)
```

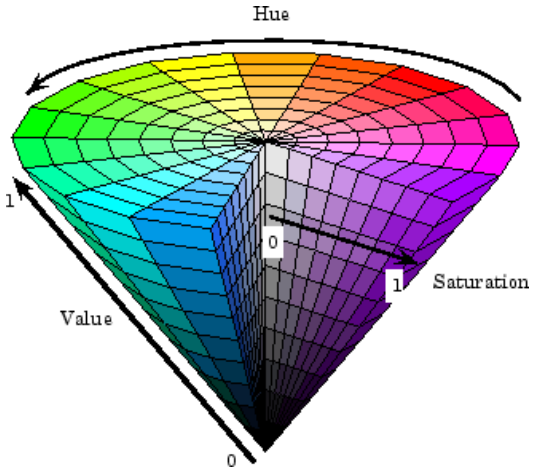
Cuerpo sigue la cabeza:

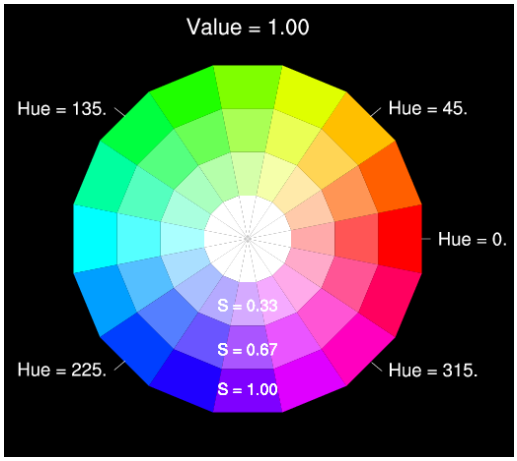
```
Speed Wheel R (cm/s) = 2*(10 + 10*headYaw.val)

Speed Wheel L (cm/s) = 2*(10 - 10*headYaw.val)
```




### 3.3.2. Identificación de los colores

Para identificar los colores, utilizamos la escala HSV puesto que es la elegida por Awabot. Sin embargo, emplean su propia escala por lo que no podemos emplear los valores HSV tradicionales.




Existe un módulo para identificar colores, pero requiere la interacción del usuario para definir los colores por lo que creamos uno propio por script. Con nuestro código, definimos los colores de antemano:

```
linkManager.registerModule(colorDetection,0); // activa la detección del color
linkManager.setVisibleModuleView(colorDetection); //met le color detection comme  vue
                                                                    par defaut
colorDetection.addColor(UColor.new, "red1");//crea la variable red1
colorDetection.setColorSpace("red1", 137, 153, 90, 256, 54, 256); //color violeta de una
                                                                    pelota en el mundo virtual
colorDetection.startDetection;
```

Para conocer los parámetros de los colores que queremos codificar, utilizamos la consola debug. Para ello escribimos `EmoxSoft.exe –dev` en la consola. De ese modo, con la ayuda del módulo Color Detection podemos determinar los valores HSV de Awabot de los colores puestos delante de la cámara del robot.

# 4. Conclusión

Gracias al proyecto hemos tenido la ocasión de conocer dos robots diferentes, diversas formas para controlarlos y, además, nuevos lenguajes de programación.

Hemos establecido las bases para el nuevo laboratorio. Con Emox, hemos hecho un gran estudio de sus características con el fin de que los nuevos alumnos tengan una mayor facilidad a la hora de usarlo y explicando la solución a los problemas más comunes.

## BIBLIOGRAFÍA


[1] http://developer.aldebaran-robotics.com
[2] http://www.youtube.com/watch?v=4eBqVn3uS5w
[3] http://searchfortechnology.com/emox-intelligent-entertainment-robot
[4] http://awabot-redmine.herokuapp.com/
[5] http://www.aldebaran-robotics.com/documentation/dev/cpp/install_guide.html
[6] http://old-releases.ubuntu.com/releases/
[7] http://code.google.com/p/qrdecoder/
[8] http://qrworld.wordpress.com/2011/09/26/qr-codes-versus-data-matrix/
[9] http://datamatrix.kaywa.com/
[10] http://www.barcodeisland.com
[11] http://www.unitaglive.com/qrcode
[12] Athanasia LOULOUDI, Ahmed MOSALLAM, Naresh MARTURI, Pieter JANSE and Victor HERNANDEZ. http://ebookbrowse.com/sais2010-ipw-paper-pdf-d79389844. School of Science and Technology. Örebro University, Sweden
[13] http://code.google.com/p/qrdecoder/
[14] http://www.riverbankcomputing.co.uk/software/pyqt/intro
[15] http://www.riverbankcomputing.co.uk/software/sip/intro
[16] http://www.pythonware.com/products/pil/
[17] http://www.pythonware.com/library/pil/handbook/image.htm
[18] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_images_using_the_API
[19] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=HOWTO:_Scan_video_using_the_Processor
[20] http://stackoverflow.com/questions/9190687/using-OpenCV-instead-of-imagemagick-in-ZBar-qr-code-decoder
[21] http://sourceforge.net/apps/mediawiki/ZBar/index.php?title=ZBar_Wiki
[22] Site officiel d'OpenCV : http://docs.OpenCV.org/
[23] http://OpenCV.willowgarage.com/wiki/
[24] http://www.mattmontag.com/development/notes-on-using-OpenCV-2-3-with-visual-studio-2010
[25] http://docs.OpenCV.org/modules/core/doc/intro.html
[26] http://www.libdmtx.org/
[27] http://imagemagick.org
[28] ZXing http://code.google.com/p/zxing/
[29] Libdecodeqr : https://github.com/josephholsten/libdecodeqr
[30] https://developer.aldebaran-robotics.com/home/

[31] http://www.aldebaran-robotics.com/documentation/software/choregraphe/index.html

EMOX

[32] http://fr.wikipedia.org/wiki/Pont_en_H
[33] http://es.wikipedia.org/wiki/Puente_H_(electronica)
[34] http://www.dema.net/pdf/sharp/GP2Y0A41SK0F.pdf
[35] http://www.alldatasheet.com/view.jsp?Searchword=GP2Y0) :
[36] http://www.pololu.com/file/0J507/HD-1581HB.pdf )
[37] http://datasheet.octopart.com/HN-GH12-1634T-R-Jameco-Reliapro-datasheet-7274139.pdf
[38] http://pdf1.alldatasheet.com/datasheet-pdf/view/42773/SHARP/GP1A53HR.html
[39] http://www.datasheetdir.com/OV530-B49+download
[40] http://www.arduino.cc/fr/
[41] http://www.jameco.com/Jameco/Products/ProdImag/161382.jpg
[42] FAGUET, Thomas. Documentation d'EMOX : *Doc Hardware.pdf.* Version 1 du 12/06/2012. Awabot Robotics.
[43] Documentation d'EMOX : *Manuel d'utilisation.pdf.* Awabot Robotics. 2012.
[44] Documentation d'EMOX : *Document de programmation graphique V1.pdf.* Awabot Robotics. Version 1.0.0. Septembre 2012.
[45] Documentation d'EMOX : *Emox_SDK.pdf.* Awabot Robotics . 18 juin 2012.
[46] http://botbench.com/blog/2011/09/16/exposed-lego-colour-sensor/
[47] http://awabot-redmine.herokuapp.com/boards/5/topics/4
[48] play.google.com/store/apps/details?id=com.mobialia.colordetector
[49] play.google.com/store/apps/details?id=com.loomatix.colorgrab
[50] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html